Studying onboarding to improve program comprehension tool support

Rebecca Yates
Lero, University of Limerick
E-mail: rebecca.yates@lero.ie

Abstract

Gaining an understanding of unfamiliar software systems is hard. Existing support tools are based on studies of the information sought by software developers, but often the developers themselves do not know what to look for. Here an alternative is proposed: by studying the information 'pushed' from software experts to new developers during onboarding, the information provided by the experts can be analysed and compared to the information 'pulled' by the new developers. The content and presentation of this data will inform tool design for onboarding support.

1. Introduction

An experienced software developer needing to understand an unfamiliar codebase may receive guidance from co-located experts. However, experts may be unavailable [3] (for example, attending a long meeting, or working in a different timezone or for a different company) and the developer is left to puzzle out the code alone. Program comprehension is a difficult task that is not well supported by current tools [7]. This is a barrier to participation in open source development [10] and a source of frustration to commercial developers [4]. Improved tools and techniques would support developers in the absence of system experts.

2. Related work

Even after many years of experience, software developers struggle to comprehend unfamiliar code [4]. It may take four or five months [8] before mentored new developers are able to work independently.

When experts are available, their help is invaluable. Studies of onboarding [8][1] show experts' roles as mentors, passing on a wealth of social and technical information that would otherwise be difficult to obtain. Experts are able to mentor without training, using highly interactive conversation to inform the new developer and check their understanding.

However, sometimes experts are unavailable [3]. In this case, developers may need to investigate a broad cross-section of code before they can make successful changes to the system [6]. They may also be unsure of how to approach maintenance tasks and unaware of other developers' knowledge of the system.

Some tool support for unguided exploration is available, and is often based or evaluated on theories of program comprehension (e.g. [9][11]). Evidence for these theories includes experiments and case studies. For the results to generalise to commercial software development, the studies must be realistic. However, comprehension experiments often recruit students instead of professional developers, or involve unrealistic tasks on small codebases, and so the results may hold true only under those conditions. Case studies and realistic experiments (e.g. [5]) provide more reliable data for understanding developers' practices and requirements [2].

In addition, both of the approaches described here only study the information 'pulled' from unfamiliar code by developers. There are no known studies of the information 'pushed' to software developers by experts, and so this information is unavailable for informing tool design.

3. Proposed study

A series of real-world case studies is proposed. Each study involves going into a company to record genuine onboarding sessions. The experimental setup is similar to [6] but with screen capture in addition to audiovisual recording, so that references to on-screen text can be clearly understood. The setup has been tested in a mocked-up onboarding scenario.

This data will be transcribed, coded and analysed to answer the following questions:

- What information about the code's design is passed from experts to onboarders?
- What methods of presentation are used to pass on this information?

- How much of the information passed on was available in the code or documentation, and how much is kept solely 'in people's heads'? [4]
- What are the differences between information sought by onboarders and provided to them by experts?

Collection of this data is difficult to arrange; this presents a risk to the study. Alternative sources of data include student programming projects and open source projects; however, each has its drawbacks and the conclusions may not generalise to commercial software development.

The second stage of the study is to use these findings to inform tool design for onboarding situations:

- Can we automatically extract any of this information from source code?
- Is the experts' presentation of the data appropriate for the onboarder?
- Would a tool based on these findings support developers during onboarding when experts are unavailable?

Studying real-world onboarding situations will provide the data to answer these questions and contribute to our understanding of the requirements for good software developement tools.

4. Contributions

Studying the information 'pushed' from experts to newcomers is a novel approach to improving tools to support onboarding. Real-world, empirical data is a strong foundation for designing tools and ensures that they address relevant issues for software developers.

There are no studies recording the order in which experts teach parts of their system. Collecting and interpreting this data to inform visualisation design will be a novel contribution.

Effective software visualisation is an ongoing research area. Evidence of real-world production and use of diagrams to explain software will feed into the discussion on effective ways to visualise software, and the suitability (or otherwise) of existing software development artifacts.

5. Conclusions

Mentoring provides valuable social as well as technical support, and should not be replaced even if that were possible. However, tools that support the comprehension of unfamiliar code would provide support to the developer when expert help is unavailable.

This paper outlines a proposed study into the information 'pushed' from software experts to new developers. The data collected from the case studies will complement existing studies of information 'pulled' by developers from the code. This is intended to lead towards the development of better tools for program comprehension.

6. Acknowledgements

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie). The author thanks Jim Buckley for his suggestions and advice.

References

[1] L. Berlin. Beyond program understanding: A look at programming expertise in industry. In *Empirical Studies of Programmers: Fifth Workshop*, pages 6–25, December 1993.

[2] J. Buckley. Requirements-based visualization tools for software maintenance and evolution. *IEEE Computer*, 42(4):106–108, April 2009.

[3] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 344–353, Washington, DC, USA, 2007. IEEE Computer Society.

[4] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: a study of developer work habits. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 492–501, New York, NY, USA, 2006. ACM.

[5] M. O'Brien and J. Buckley. Modelling the information-seeking behaviour of programmers - an empirical approach. In *Proceedings of the 13th International Workshop on Program Comprehension*, pages 125–134, May 2005.

[6] M. P. Robillard, W. Coelho, and G. C. Murphy. How effective developers investigate source code: An exploratory study. *IEEE Transactions on Software Engineering*, 30(12):889–903, December 2004.

[7] J. Sillito, G. Murphy, and K. De Volder. Asking and answering questions during a programming change task. *IEEE Transactions on Software Engineering*, 34:434–451, 2008.

[8] S. Sim and R. Holt. The ramp-up problem in software projects: a case study of how software immigrants naturalize. In *Software Engineering Proceedings of the 1998 International Conference on*, pages 361–370, Apr 1998.

[9] V. Sinha, D. Karger, and R. Miller. Relo: helping users manage context during interactive exploratory visualization of large codebases. In *eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 21–25, New York, NY, USA, 2005. ACM.

[10] G. von Krogh, S. Spaeth, and K. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, 2003.

[11] A. von Mayrhauser and A. M. Vans. From program comprehension to tool requirements for an industrial environment. In *Proceedings of IEEE Workshop on Program Comprehension*, pages 78–86. IEEE Comp. Soc. Press, 1993.